Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина «Избранные главы информатики»

**ОТЧЕТ**

к лабораторной работе №2

на тему:

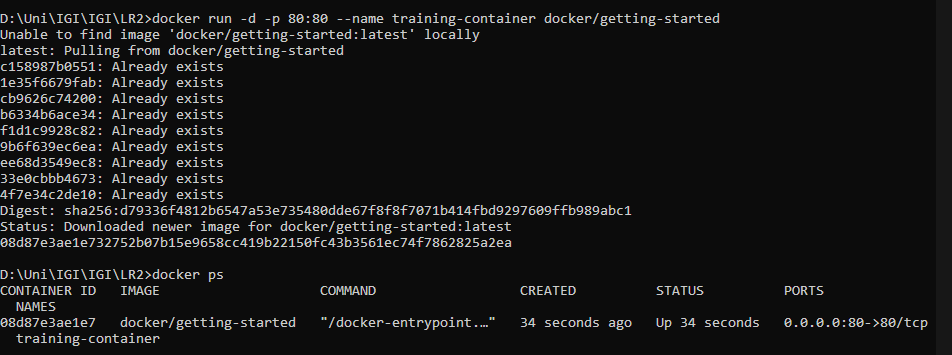
**«Docker»**

БГУИР 1-40-04-01

|  |
| --- |
| Выполнил студент группы 253504  Дмитрук Богдан Ярославович |
| 09.03.2023 |
| (дата, подпись студента) |
| Проверила  Жвакина Анна Васильевна |
|  |
| (дата, подпись преподавателя) |

Минск 2023

1. Изучите простейшие консольные команды и возможности Docker Desktop (см. лекцию), создать собственный контейнер docker/getting-started, открыть в браузере и изучить tutorial.



1. Создайте docker image, который запускает скрипт с использованием функций из https://github.com/smartiqaorg/geometric\_lib.
2. Данные необходимые для работы скрипта передайте любым удобным способом (например: конфиг файл через docker volume, переменные окружения, перенаправление ввода). Изучите простейшие консольные команды для работы с docker(см. лекцию). Зарегистрируйтесь на DockerHub и выберите необходимые для проекта образы
3. Создать Dockerfile для реализации сборки собственных Docker образов
4. Использовать его для создания контейнера. Протестировать использование контейнера
5. Скачать любой доступный проект с GitHub с произвольным стеком технологий или использовать свой, ранее разработанный. Создать для него необходимый контейнер, используя Docker Compose для управления многоконтейнерными приложениями. Запустить проект в контейнере.( Примеры Images: https://hub.docker.com/\_/phpmyadmin, https://hub.docker.com/\_/mysql, https://hub.docker.com/\_/postgres)
6. Настроить сети и тома для обеспечения связи между контейнерами и сохранения данных (исходные данные, логин, пароль и т.д.)
7. Разместите результат в созданный репозиторий в DockerHub
8. Выполните следующие действия с целью изучить особенности сетевого взаимодействия:
   * Получить информацию о всех сетях, работающих на текущем хосте и подробности о каждом типе сети
   * Создать свою собственную сеть bridge, проверить, создана ли она, запустить Docker-контейнер в созданной сети, вывести о ней всю информацию(включая IP-адрес контейнера), отключить сеть от контейнера
9. • Создать еще одну сеть bridge, вывести о ней всю информацию, запустить в ней три контейнера, подключиться к любому из контейнеров и пропинговать два других из оболочки контейнера, убедиться, что между контейнерами происходит общение по IP-адресу
   * Создать свою собственную сеть overlay, проверить, создана ли она, вывести о ней всю информацию
   * Создать еще одну сеть overlay, проверить, создана ли она, вывести о ней всю информацию, удалить сеть
   * Попробовать создать сеть host, сохранить результат в отчет.

Теория к сдаче

1. **docker build –** docker build -t <image\_name>:<tag> <path\_to\_dockerfile> — используется для создания образа Docker из Dockerfile

EX: docker build -t myimage:1.0 . - создание образа с именем "myimage" и тегом "1.0" на основе Dockerfile, находящегося в текущем каталоге.

1. **docker tag --** docker tag <existing\_image\_name>:<tag> <new\_image\_name>:<new\_tag> — используется для создания новой метки (тега) для существующего образа Docker.

EX: docker tag myimage:1.0 myimage:latest — создание новой метку latest для существующего образа myimage с тегом 1.0. Теперь у нас есть две метки (1.0 и latest), указывающие на один и тот же образ.

1. **docker run --** docker run [flags] <image\_name> [command] – создание и запуск контейнера на основе определенного образа.

[command] - опциональная команда, которая будет выполнена внутри контейнера при его запуске. Если команда не указана, будет выполнена команда, заданная внутри образа по умолчанию.

EX: Запуск контейнера с указанием порта хоста и порта контейнера

docker run -p 8080:80 myimage

1. **docker start** указал идентификатор и остановленный контейнер запустился
2. **docker stop** указал идентификатор и запущенный контейнер остановился
3. **docker pause** указал идентификатор и запущенный контейнер стал на паузу
4. **docker unpause** указал идентификатор и контейнер на паузе опять дышит
5. **docker restart**



1. **docker ps –** показывает все запущенные контейнеры
2. **docker logs –** используется для просмотра логов, сгенерированных контейнером
3. **docker images**



1. **docker network**

В контексте Docker, сеть представляет собой виртуальное средство связи, которое позволяет контейнерам Docker общаться друг с другом и с внешними ресурсами, такими как хостовая машина или другие сетевые сервисы.

Сеть Docker может быть создана для группировки и изоляции контейнеров, чтобы они могли обмениваться данными и взаимодействовать друг с другом. Когда контейнеры находятся в одной сети, они могут использовать DNS для обнаружения других контейнеров по их именам и общаться между собой с помощью сетевых протоколов.

Сеть Docker также может предоставлять контейнерам доступ к внешним ресурсам или сетевым сервисам, таким как базы данных, веб-серверы или другие контейнеры. Контейнеры, подключенные к одной сети, могут общаться с любыми другими контейнерами в этой сети, независимо от того, находятся ли они на одной хостовой машине или на разных.

Usage: docker network COMMAND

Commands:

connect <network> <container> Connect a container to a network

create <network> -- Create a network

disconnect <network> <container> Disconnect a container from a network

inspect <network> -- Display detailed information on one or more networks

ls List networks

prune Remove all unused networks

rm Remove one or more networks. All containers connected to that network will be disconnected from it.

1. **docker volume**

Volume в контексте Docker - это механизм для сохранения и управления данными, используемыми контейнерами Docker. Volume представляет собой отдельное хранилище данных, которое может быть присоединено к одному или нескольким контейнерам.

Основная цель использования Docker volume состоит в том, чтобы обеспечить постоянное хранение данных, которые нужны контейнеру, даже после его удаления или перезапуска. Это позволяет сохранить состояние приложения или обмениваться данными между контейнерами.

EX: docker run -v myvolume:/path/to/mount myimage  
Создание volume с именем myvolume, а затем контейнер запускается с присоединением этого volume к пути /path/to/mount внутри контейнера.

Usage: docker volume COMMAND

Commands:

сreate Create a volume

inspect Display detailed information on one or more volumes

ls List volumes

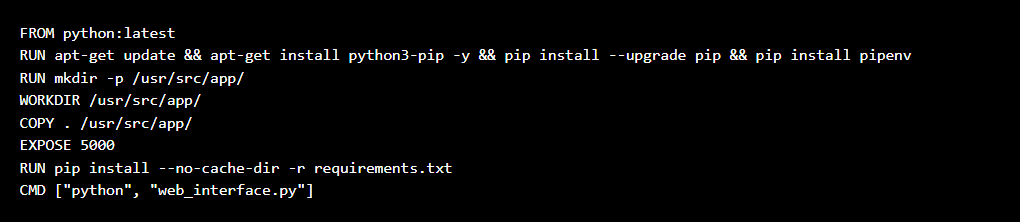
prune Remove unused local volumes

rm Remove one or more volumes

1. **docker inspect <** **object name > —** используется для получения подробной информации о Docker-объекте, таком как контейнер, образ, сеть или том. Она предоставляет доступ к метаданным и настройкам объекта в формате JSON.
2. **Dockerfile**

**Dockerfile** — это конфигурационный файл, в котором описаны инструкции, которые будут применены при сборке Docker-образа и запуске контейнера. Dockerfile создается в корневой директории проекта и не имеет расширения. Инструкции пишутся капсом, а их значения отделяются пробелом.

Приведем пример несложного Dockerfile, и на его примере разберем логику взаимодействия инструкций между собой.



Dockerfile имеет следующую логику заполнения:

1. Первой инструкцией всегда идёт FROM с указанием родительского образа. Например, FROM python:latest.
2. Инструкция RUN может принимать конвейер команд Linux, чтобы не создавать лишние слои. Например, RUN apt-get update && apt-get install python3-pip -y && pip install --upgrade pip && pip install pipenv.
3. Инструкция WORKDIR устанавливает рабочий каталог контейнера. Например, WORKDIR /usr/src/app/. Последующие команды RUN, CMD, ENTRYPOINT наследуют привязку WORKDIR.
4. COPY в Dockerfile используется для копирования файлов и директорий из локальной файловой системы в образ Docker, который вы создаете. COPY <source> <destination>
5. Инструкция EXPOSE, которая выражает намерение открыть заданный порт. Инструкция сама по себе не открывает порт без применения команды docker run с ключом -P. Если нужно «повесить» контейнер на определённый внешний порт с переадресацией во внутренний порт контейнера — применяется ключ -p с указанием внутреннего и внешнего порта через «:». Например, docker run -p 5000:8080
6. Завершающей инструкцией всегда идёт CMD. Например, CMD ["python", "web\_interface.py"]. CMD наследует привязку к WORKDIR, поэтому web\_interface.py будет запущен из папки /usr/src/app/.

| **Инструкция** | **Описание** | **Пример использования** | **Комментарий** |
| --- | --- | --- | --- |
| FROM | Задает базовый образ. Все последующие инструкции создают слои поверх родительского образа. | FROM python:latest FROM debian:wheezy | Быстрее всего можно найти образ с нужным тегом на [Docker hub](https://hub.docker.com/). |
| RUN | Выполняет команду внутри контейнера и сохраняет результат. | RUN mkdir /usr/src/app/ RUN apt-get update && apt-get install python3-pip -y | RUN может исполнять конвейер команд с логическими операторами && и ||. |
| COPY | Копирует файлы и папки из текущей директории, где находится пользователь в указанную директорию в контейнере | COPY . /usr/src/app/ | COPY считывает позицию пользователя на хосте, поэтому первым аргументом идет «.». |
| ADD | Копирует файлы и папки из текущей позиции пользователя, скачивает файлы по URL и работает с tar-архивами. | ADD https://1cloud.ru/archive/api\_config.ini /usr/src/app/ | Официальная документация не рекомендует применять ADD. Для скачивания по URL можно использовать RUN с CURL или WGET, а для копирования — COPY. |
| CMD | Выполняет команду с указанными аргументами во время запуска контейнера. | CMD ["python", "web\_interface.py"] | CMD должна быть одна в конце Dockerfile. CMD может вызывать исполняемый файл — .sh Аргументы docker run переопределяют CMD. Если в Dockerfile нет CMD, обязательно должна быть инструкция ENTRYPOINT. |
| ENTRYPOINT | Похожа на CMD, но при запуске контейнера не переопределяется в отличие от CMD. | ENTRYPOINT ["python", "web\_interface.py"] | ENTRYPOINT может использоваться совместно с CMD. |
| ENV | Задает переменные среды внутри образа, на которые могут ссылаться другие инструкции. | ENV ADMIN="ivan" | ENV часто применяется для передачи информации в контейнеризированное приложение через переменные среды. |
| ARG | Задает переменные, значение которых передается докером во время сборки образа. | ARG maintainer=ivan | В отличие от ENV-переменных, ARG-переменные недоступны во время выполнения контейнера. |
| WORKDIR | Устанавливает рабочую директорию контейнера. | WORKDIR /usr/src/app/ | Последующие инструкции CMD, RUN, ENTRYPOINT наследуют привязку к директории установленной WORKDIR. |
| VOLUME | Создает и подключает постоянный том хранения данных. | VOLUME /data\_cont\_1 | Просмотреть существующие тома можно командой docker volume ls. К контейнеру можно подключить существующий том, для этого достаточно указать уже существующий том. |
| EXPOSE | Указывает планируемый рабочий порт у контейнера. Инструкция сама по себе не открывает порт. Чтобы использовался указанный в EXPOSE порт — нужно указать docker run -P при запуске контейнера. | EXPOSE 5000 | Если требуется пробросить и сопоставить разные порты внутри и снаружи контейнера используется docker run -p внутренний порт:внешний порт |
| LABEL | Добавляет метаданные в образ. | LABEL maintainer="katkov\_ivan" | Обычно LABEL содержит информацию об авторе образа. |

1. **Docker Compose**— инструмент, разработанный для помощи в определении и совместном использовании многоконтейнерных приложений. Используется файл .yml для определения сервисов**.**

Пример такого прикола:

version: '3'

services:

  web:

    build:

      context: .

      dockerfile: Dockerfile

    ports:

      - 80:80

    volumes:

      - ./app:/app

    depends\_on:

      - db

  db:

    image: mysql:5.7

    environment:

      - MYSQL\_ROOT\_PASSWORD=secret

      - MYSQL\_DATABASE=myapp

      - MYSQL\_USER=user

      - MYSQL\_PASSWORD=password

    volumes:

      - dbdata:/var/lib/mysql

volumes:

  dbdata:

В этом примере у нас есть два сервиса: web и db.

Сервис web собирается с использованием Dockerfile, который находится в текущем контексте (текущая директория) и имеет имя Dockerfile. Он проксирует порт 80 хостовой машины на порт 80 контейнера. Каталог ./app с локальной файловой системы монтируется внутрь контейнера по пути /app. Также web зависит от сервиса db.

Сервис db использует образ MySQL версии 5.7. Он устанавливает несколько переменных среды, включая пароль для root пользователя и создает базу данных и пользователя. Данные MySQL сохраняются в томе dbdata.

Под "сервисом" в контексте Docker Compose подразумевается отдельный контейнер Docker, который выполняет определенную функцию в вашем приложении. Каждый сервис в docker-compose.yml представляет собой отдельный экземпляр контейнера Docker, который может содержать свои настройки, зависимости, порты и другие параметры.